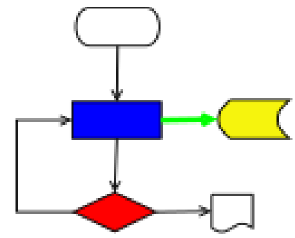
VICI



VISUAL CHART INTERPRETER

Design for libsearch

Publication History

Date	Who	What Changes
4 October 2012	Brenton Ross	Initial version.
4 January 2014	Brenton Ross	Detailed Design
1 July 2014	Brenton Ross	Updated for Vici

Copyright © 2009 - 2014 Brenton Ross
This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.
The software is released under the terms of the GNU General Public License version 3.

Table of Contents

1 Introduction......5
1.1 Scope......5
1.2 Overview......5
1.3 Audience......5
2 Overview......6
2.1 Responsibilities......6
2.2 Interfaces......6
2.2.1 IF08 Search UI......6
2.2.2 IF22 Vici-ed using Search......6
2.2.3 IF33 Search using libcfi......7
2.2.4 IF34 Search using libQtGui......7
2.3 Design Approach......7
3 User Interface......8
4 Application Design......10
5 Collaboration Diagrams......11
5.1 Application Start......11
5.2 Application Close......12
6 Class Designs......13
6.1 VennButton Class......13
6.2 CmndXml Class......13
6.3 TagXml Class......13
6.4 ClassNode Class......14
6.5 ClassTree Class......14
6.6 TreeItem Class......15
6.7 TreeModel Class......15
6.8 TreeView Class......16
6.9 SearchTagsItem Class......16
6.10 SearchTagsModel Class......17
6.11 SearchTagsView Class......17
6.12 TagValidator Class......18
6.13 SystemSearcher Class......18
6.14 ProcessSearcher Class......18
6.15 ViciSearcher Class......19
6.16 ManSearcher Class......19
6.17 DesktopSearcher Class......19
6.18 PackageSearcher Class......19
6.19 WhatIs Class......20
6.20 VennMgr Class......20
6.21 ProgItem Class......20
6.22 SearchUI Class......21
6.23 SearchFactoryImpl Class......21
6.24 SearchImpl Class......21
Appendix A......22

1 Introduction

This is part of the system design document for the VICI project.

1.1 Scope

This document covers the detailed design of the search component. This component is responsible for allowing the user to easily find commands that suit their purpose.

The document will cover the Application Design and the User Interface Design.

This design is for increment #4.

1.2 Overview

The detailed design includes:

- Interface Stubs: A framework of facade classes for the modules.
- Use Case Descriptions: A description of how a user is expected to interact with the application.
- Application Design: The classes and their relationships.
- User Interface Design: The design and layout of the graphical components of the system.
- Persistent Storage Design: The specifications for the XML files used to store configuration and scripts.

1.3 Audience

This document is intended to be used by the designers and developers, and later the maintainers, of the VICI project.

2 Overview

2.1 Responsibilities

This component allows the user to find commands for whatever task they have in mind. It addresses the following responsibilities:

T11.1: Key word search over the help text of prepared commands.

T11.2: Use the apropos command to search for commands using a key word.

T12.1: Adding, editing and removing commands from the tag database.

T12.2: Adding, editing and removing tags from the tag database.

T12.3: Classifying commands and tags as members of other tags.

T12.4: Saving and loading the tag database.

T12.6: Construct a query based on union and intersection of tags.

T12.7: Display list of commands that satisfy a query.

2.2 Interfaces

The architecture document describes the following interfaces to the search library. The library must implement these interfaces.

2.2.1 IF08 Search UI

This is the user interface that allows a user to search for a command.

Transport Medium: Displayed in a window.

Protocol: Event driven with Windows, Icons, Menus and a Pointer.

Content:

1. Search key (I)
2. Search method (I)
3. Search results (O)
4. Command selection (I)
5. Tag name (I)
6. Classification (I)
7. Tag selection (I)

2.2.2 IF22 Vici-ed using Search

This is the interface the vici-ed uses to display the search component.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Sub-window in which the search component operates. (I)
2. Selected command. (O)

2.2.3 IF33 Search using libcfi

This is the interface that Search uses to read and save the tag database.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Create an XML document structure.
2. Add nodes to the document.
3. Add properties to nodes.
4. Save the document.
5. Read an XML file
6. Get nodes from the document
7. Remove nodes from the document.
8. Get properties from the document node.

2.2.4 IF34 Search using libQtGui

This interface allows the search component to interact with the user.

Transport Medium: Memory

Protocol: C++ function calls.

Content:
1. Lists of commands and tags. (I)
2. User selections (O)

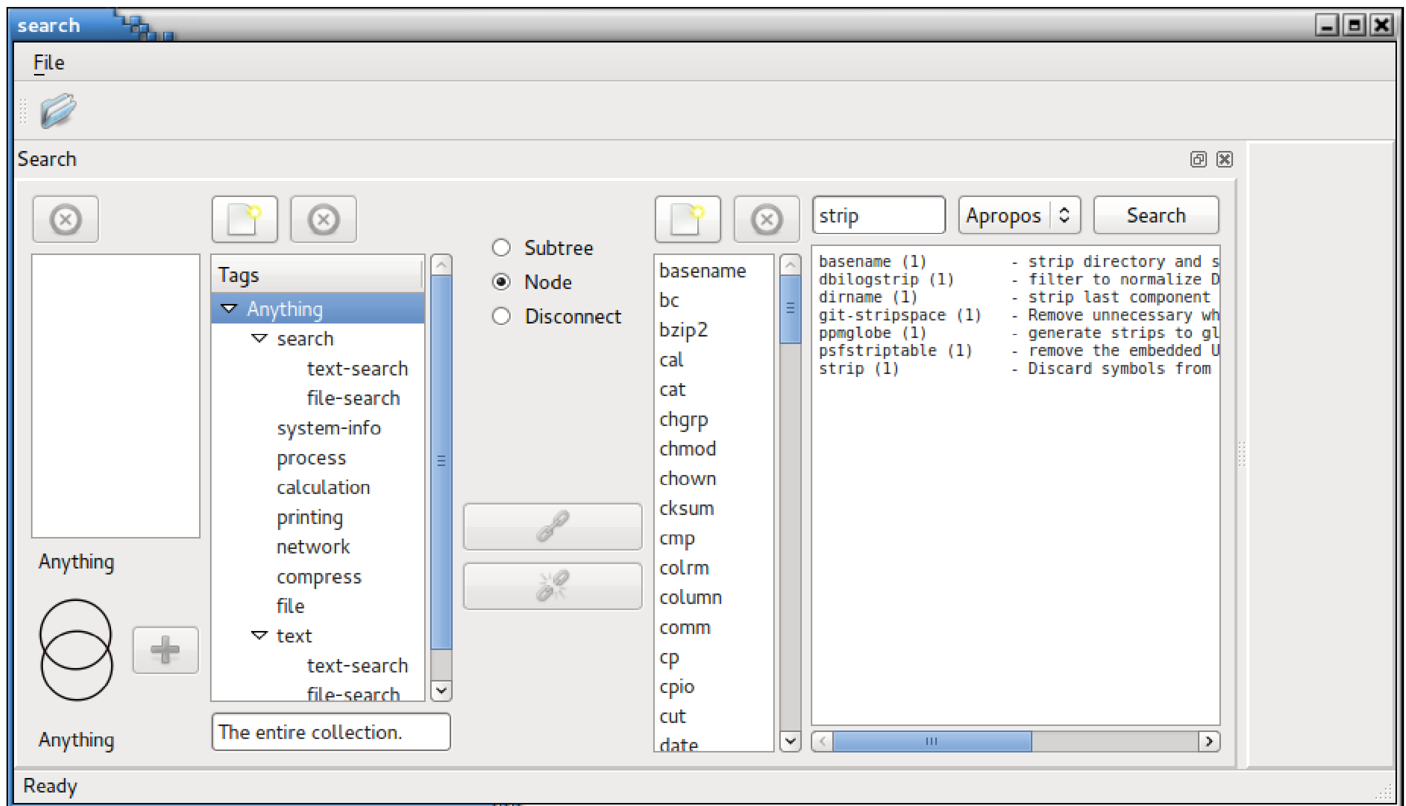
2.3 Design Approach

The Qt library provides a suitable set of widgets for building this user interface component. A feature of the Qt library is an extension to the class interface that allows one to define functions as “signals” or “slots” which can then be connected together making the usual widget call-back pattern unnecessary.

The most complex part of this component is the requirement for drag-n-drop interactions within the tag tree and between the tag tree and the list of sets.

3 User Interface

The following diagram illustrates the user interface for the search component.



At the top right the user can enter a search expression (a regular expression) and specify the area to be searched (the Vici database, man pages, desktop programs, or installed packages). The results are displayed in the right hand panel.

The list, next to the left, shows the programs. This will be initialised to the Vici prepared commands, but the user can add (or remove) other commands that they want to classify. The associated add and remove buttons are enabled when the “Anything” tag is selected since all programs must be associated with this tag.

The buttons in the middle panel allow the user to control the relationships between the tags and the programs.

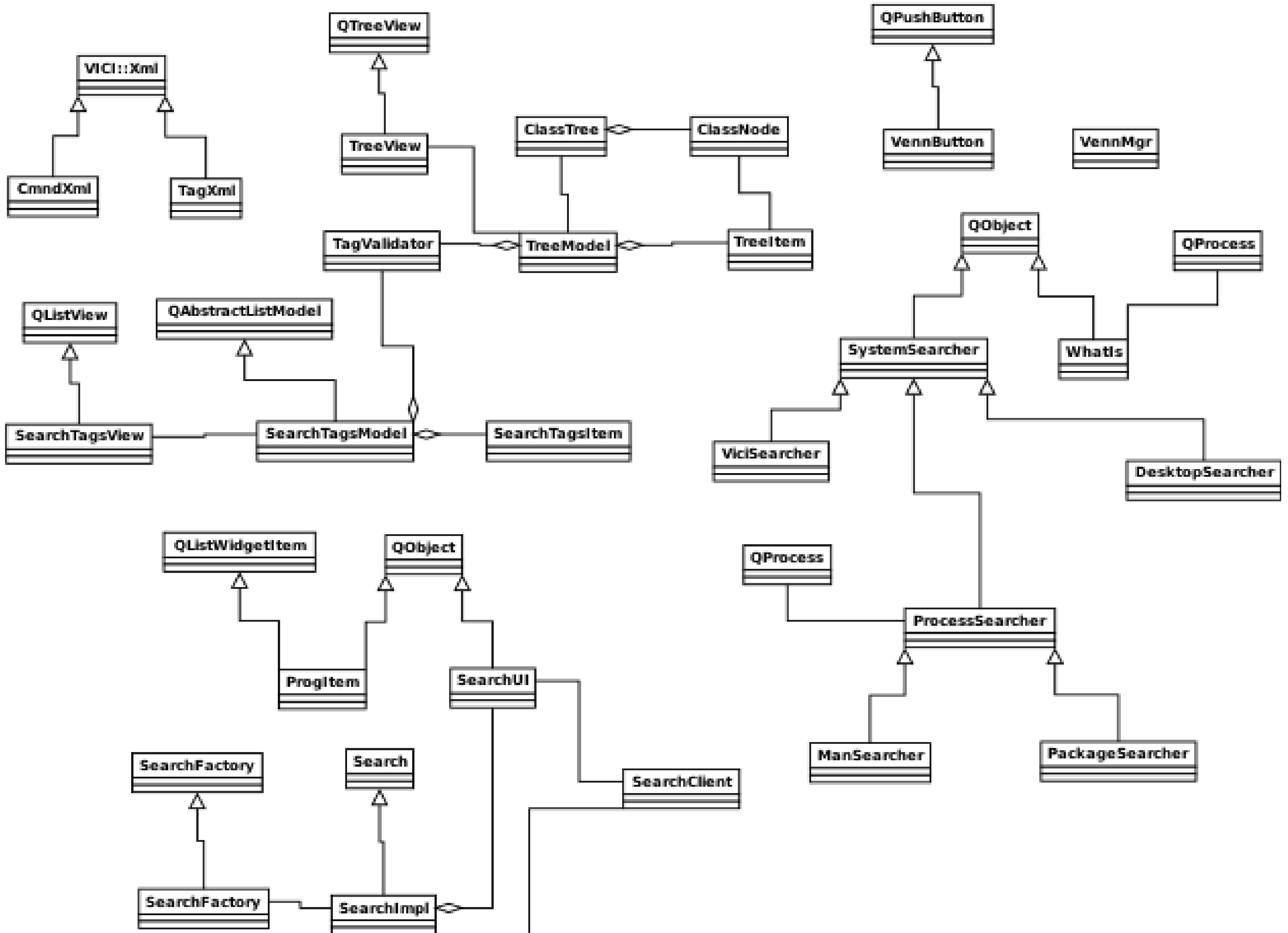
The Tags panel shows a tree of the tags. Note that a tag can have multiple parents in the tree. (Note that “text-search” is a child of both “text” and “search” in the illustration.) Tags can be dragged and dropped within the tree to either move or copy them. (Hold the shift key while dragging to perform a move.)

The left hand panel is a list of named sets. These are created using the Venn

diagram at the lower left. The labels above and below the Venn diagram refer to the last two tags or sets that have been selected. The user can then create a new set by clicking within the circles of the Venn diagram. Pressing the Add button will then create a new set which can be named. These named sets can be dragged onto the tag tree to become new tags.

4 Application Design

The following diagram describes the relationships between the classes used in this library.

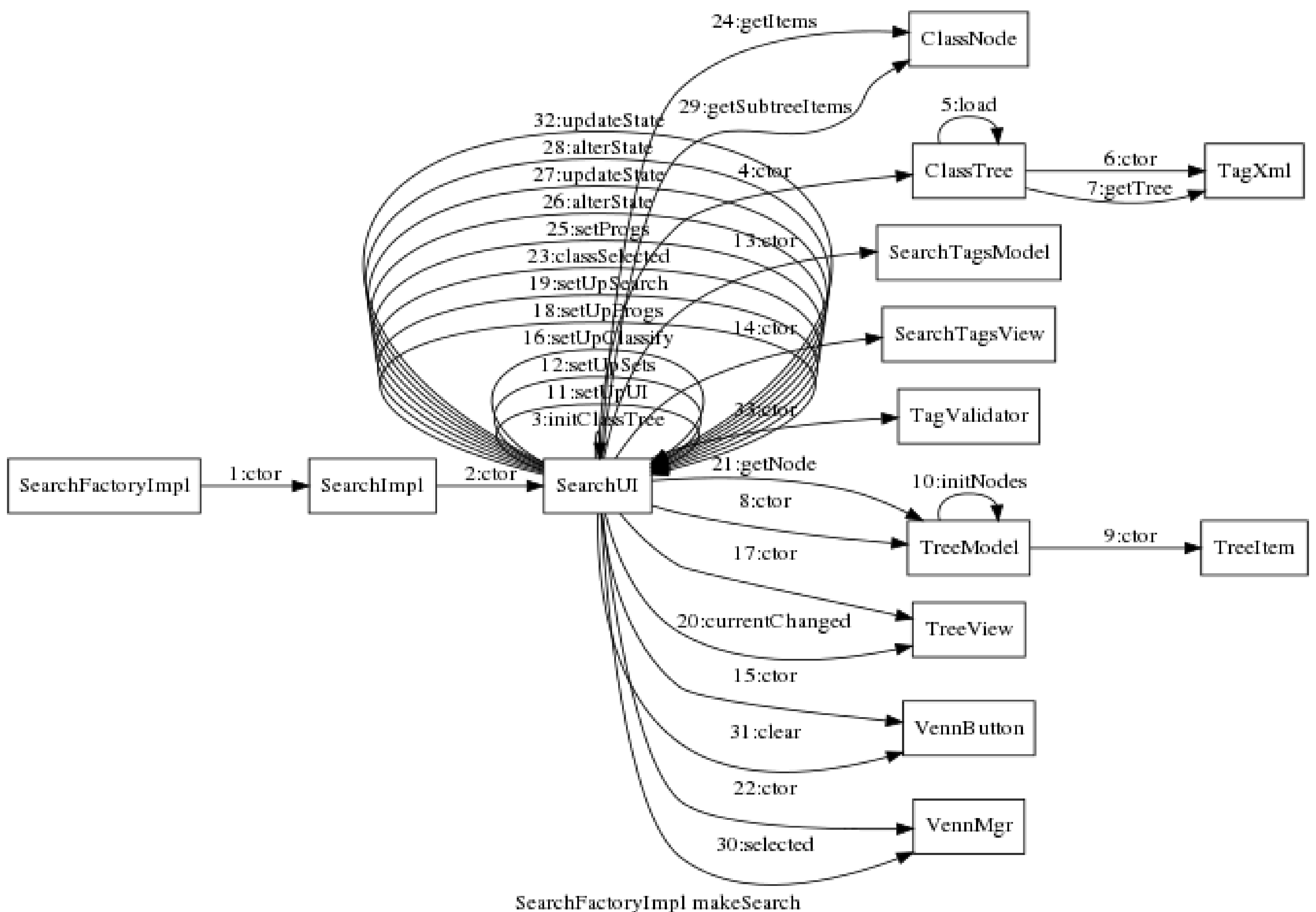


5 Collaboration Diagrams

The following diagrams illustrate the interactions that take place when use cases are performed.

5.1 Application Start

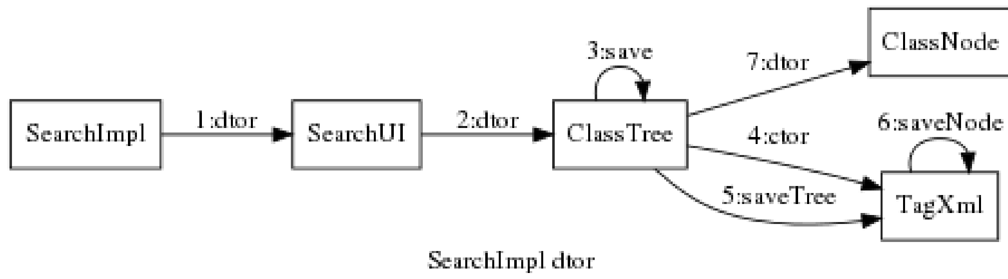
The following diagram describes the sequence of calls when the search library starts up.



SearchFactoryImpl makeSearch

5.2 Application Close

The following sequence of calls occur during shut down.



SearchImpl dtor

6 Class Designs

This section describes each class, including its responsibilities, and its public and protected members.

6.1 VennButton Class

This class is responsible for presenting a clickable Venn diagram.

```
class VennButton : public QPushButton
{
  signals:
        void hit( const QPoint & pos ) const;
        void changed( int ) const;
public:
        VennButton();
        ~VennButton();
        virtual bool hitButton ( const QPoint & pos ) const;
        bool vennState() const { return vstate; }
        void clear();
};
```

6.2 CmndXml Class

This class is responsible for getting the command descriptions from the Vici command database.

```
class  CmndXml : public VICI::Xml
{
public:
        CmndXml();

        void getDescriptions( std::map< QString, QString > & );
};
```

6.3 TagXml Class

This class is responsible for loading and saving the tag database.

```
class TagXml : public VICI::Xml
{
public:
        TagXml();

        void getTree( ClassTree * );
        void saveTree( ClassTree * );
};
```

6.4 ClassNode Class

This class represents a tag in the tree. The ClassNodes can have multiple parents as well as multiple children, and each is associated with a set of programs (called items).

```
class ClassNode
{
public:
        ClassNode( VICI::csr name );
        ~ClassNode();
        void unlink( ClassNode * );     // remove a child
        void dump( int indent );
        void addItems( const std::vector< std::string > & );
        void delItems( const std::vector< std::string > & );
        void delItem( VICI::csr );  // from this and all its children
        void renameItem( VICI::csr newName, VICI::csr oldName ); // apply to children

        void getItems( std::set< std::string > & );
        void getSubtreeItems( std::set< std::string > & );

        std::string name;
        std::string description;
        std::list< ClassNode * > parents;
        std::list< ClassNode * > children;
        std::set< std::string > items;

        // not implemented - ClassNodes are not to be copied
        ClassNode( const ClassNode & );
        ClassNode & operator = ( const ClassNode & );
};
```

6.5 ClassTree Class

This class is responsible for maintaining the hierachy of ClassNode objects. It includes a map that allows nodes to be found by name.

```
class ClassTree
{
public:
        ClassTree();
        ~ClassTree();

        void load();
        void save();
        void clear();
        void delItem( VICI::csr );  // remove an item from all nodes
        void addItem( VICI::csr );     // add an item to Anything
        bool renameItem( VICI::csr curr, VICI::csr prev ); // rename on all nodes

        ClassNode *root;

        // lookup table to allow finding by name
        std::map< std::string, ClassNode * > treeNodes;
        std::string newTagName();
};
```

6.6 TreeItem Class

This class interfaces between the ClassNodes of the ClassTree and the items of the TreeModel. These objects form a single inheritance tree (i.e. they have a single parent) but can be associated with a ClassNode that has multiple parents.

```
class TreeItem
{
public:
        TreeItem( ClassNode *, TreeItem *parent = 0);
        ~TreeItem();

        // get nth child of this node
        TreeItem *child( int ) const;

        // get parent of this node
        TreeItem *parent() const;

        // get number of children
        int childCount() const;

        // get the position of this in its parent list
        int childNumber() const;

        QVariant data( int column ) const;

        ClassNode *classNode() const;

        // insert new children
        void insertChildren( int row, int count, ClassTree * );
        void removeChildren( int row, int count, ClassTree * );
        bool setData( int column, const QString &text, ClassTree * );
        void dump(int indent);
};
```

6.7 TreeModel Class

This class provides the internal representation of the data for tree widget.

```
class TreeModel : public QAbstractItemModel
{
public:
        TreeModel( ClassTree * tree );
        ~TreeModel();
        void setValidator( TagValidator *tv );

        // top of displayed tree
        QModelIndex anything() const;

        virtual Qt::ItemFlags flags( const QModelIndex & index ) const;
        virtual QVariant data( const QModelIndex &, int ) const;
        virtual int columnCount( const QModelIndex & ) const;

        // return number of rows that are the immediate children of the indexed item
        virtual int rowCount( const QModelIndex & ) const;
```

```
        virtual QModelIndex parent( const QModelIndex & ) const;
        virtual QModelIndex index( int row, int column, const QModelIndex & ) const;

        virtual QVariant headerData( int section, Qt::Orientation, int role ) const;

        virtual bool setData( const QModelIndex & index, const QVariant & value,
                                int role = Qt::EditRole );

        virtual bool insertRows( int row, int count, const QModelIndex & parent = QModelIndex() );
        virtual bool removeRows ( int row, int count, const QModelIndex & parent = QModelIndex() );
        virtual bool insertColumns ( int column, int count,
                                const QModelIndex & parent = QModelIndex() );

        virtual bool removeColumns ( int column, int count,
                                const QModelIndex & parent = QModelIndex() );

        virtual Qt::DropActions supportedDropActions() const;
        virtual bool dropMimeData(const QMimeData *data, Qt::DropAction action,
                        int row, int column, const QModelIndex &parent);
        virtual QStringList mimeTypes() const;
        virtual QMimeData *mimeData(const QModelIndexList &indexes) const;

        ClassNode *getNode( QModelIndex );

        void dump( csr );  // dump tree to stdout
};
```

6.8 TreeView Class

This class provides the view of the tree widget. It provides additional signals and slots to allow events on the tree to be responded to.

```
class TreeView : public QTreeView
{
public:
        TreeView( QWidget * parent = 0 );

protected slots:
        void currentChanged ( const QModelIndex & current, const QModelIndex & previous );

signals:
        void selectionChanged( const QModelIndex & );
};
```

6.9 SearchTagsItem Class

This class represents the items in the list of search tags.

```
class SearchTagsItem
{
public:
        SearchTagsItem();
        QString name;
        std::set< std::string > progItems;
        virtual Qt::ItemFlags flags(const QModelIndex &index) const;
};
```

6.10 SearchTagsModel Class

This class provides the internal data for our derived list widget.

```
class SearchTagsModel : public QAbstractListModel
{
public:
        SearchTagsModel(QWidget * parent = 0);
        void setValidator( TagValidator *tv );
        const SearchTagsItem & getItem( QModelIndex ) const;
        int rowCount(const QModelIndex &parent = QModelIndex()) const;
        const SearchTagsItem * find( const QString & ) const;
        QVariant data(const QModelIndex &index, int role) const;
        QVariant headerData(int section, Qt::Orientation orientation,
                        int role = Qt::DisplayRole) const;
        virtual Qt::ItemFlags flags( const QModelIndex & index ) const;
        virtual bool setData ( const QModelIndex & index, const QVariant & value,
                                int role = Qt::EditRole );
        void setData( const QModelIndex & index, const SearchTagsItem & );
        virtual bool setHeaderData ( int section, Qt::Orientation orientation,
                        const QVariant & value, int role = Qt::EditRole );
        virtual bool insertRows ( int row, int count,
                          const QModelIndex & parent = QModelIndex());
        virtual bool removeRows ( int row, int count,
                          const QModelIndex & parent = QModelIndex() );
        virtual Qt::DropActions supportedDropActions() const;
        virtual QStringList mimeTypes() const;
        virtual QMimeData *mimeData(const QModelIndexList &indexes) const;
};
```

6.11 SearchTagsView Class

This class provides the view to our search tags widget.

```
class SearchTagsView : public QListView
{
public:
        SearchTagsView( QWidget * parent = 0 );
protected slots:
        virtual void selectionChanged ( const QItemSelection &
selected,
                                const QItemSelection & deselected );
signals:
        void selectionChanged( bool isSelected );
};
```

6.12 TagValidator Class

This class is used to ensure that tag names are unique.

```
class TagValidator
{
public:
        TagValidator( ClassTree *, SearchTagsModel *);
        bool isOK( const QString & tagName );
};
```

6.13 SystemSearcher Class

This abstract class provides the common functionality for searching.

```
class SystemSearcher : public QObject
{
protected:
        QTextEdit *mDisplay;    // referenced
public:
        SystemSearcher( QTextEdit *resultDisplay )
                : mDisplay( resultDisplay ) {}
        virtual ~SystemSearcher() {}
        virtual void searchFor( const QString & ) = 0;

signals:
        void done();
};
```

6.14 ProcessSearcher Class

This class provides the common functionality for classes that use QProcess to help with searches.

```
class ProcessSearcher : public SystemSearcher
{
protected:
        QProcess *process;      // owned
        void endProcess();
        QByteArray data;
        bool finished;

public:
        ProcessSearcher( QTextEdit *resultDisplay );
        virtual ~ProcessSearcher();

protected slots:
        virtual void readFromProcess();
        void procFinished(  int exitCode,
                       QProcess::ExitStatus exitStatus );
};
```

6.15 ViciSearcher Class

This class performs searches of the Vici command database.

```
class ViciSearcher : public SystemSearcher
{
public:
        ViciSearcher( QTextEdit *resultDisplay );
        virtual void searchFor( const QString & );
};
```

6.16 ManSearcher Class

This class uses the apropos command to search for man pages for the requested pattern.

```
class ManSearcher : public ProcessSearcher
{
public:
        ManSearcher( QTextEdit *resultDisplay );
        virtual void searchFor( const QString & );
};
```

6.17 DesktopSearcher Class

This class searches for the desired pattern in the comment descriptions of the “desktop” files which define the desktop appearance of graphical programs.

(This is outside the scope of VICI, and is provided as a convenience.)

```
class DesktopSearcher : public SystemSearcher
{
public:
        DesktopSearcher( QTextEdit *resultDisplay );
        virtual void searchFor( const QString & );
};
```

6.18 PackageSearcher Class

This class searches for the desired pattern in the summaries of package descriptions in the rpm database.

(This is outside the scope of VICI and is provided as a convenience.)

```
class PackageSearcher : public ProcessSearcher
{
protected slots:
        virtual void readFromProcess();

public:
        PackageSearcher( QTextEdit *resultDisplay );
        virtual void searchFor( const QString & );
};
```

6.19 WhatIs Class

This runs the whatis command on the selected list of programs to provide the user with a description of those commands.

```
class WhatIs : public QObject
{
 public:
        WhatIs( QTextEdit *resultDisplay );
        ~WhatIs();
        void searchFor( const QStringList & );
};
```

6.20 VennMgr Class

This class manages the calculation of the sets according to the operations specified by the VennButton object.

```
class VennMgr
{
public:
        VennMgr( QLabel *upper, QLabel *lower,
                    const std::set< std::string > & );

        // called when the user selects a tag or another set
        void selected( const QString &name,
                    const std::set< std::string > & );
        void deleted( const QString &name );
        void vennChanged( int vstate );

        const std::set< std::string > & getVennSet() const;
        int getState() const;
};
```

6.21 ProgItem Class

These hold the data for the list widget holding the list of programs. It adds a signal to allow the application to be notified of changes to the items.

```
class ProgItem : public QObject, public QListWidgetItem
{
public:
        ProgItem( const QString &, QListWidget *, ClassTree * );
        void setData ( int role, const QVariant & value );
signals:
        void changeText( const QString & curr,
                    const QString & prev );
};
```

6.22 SearchUI Class

This class manages the user interactions between all the widgets.

```
class SearchUI : public QObject
{
public slots:
        void searchNameChanged( const QString & text );
        void runSearch();
        void searchDone();
        void addProgram();
        void delProgram();
        void progChanged();
        void progEdited( const QString & curr,
                         const QString & prev );
        void classSelected( const QModelIndex &);
        void classBtnClicked( int );
        void classDescrEdited();
        void addTag();
        void delTag();
        void addAssociations();
        void delAssociations();
        void vennChanged( int );
        void addSet();
        void delSet();
        void setPressed( QModelIndex );
        void setSelectionChanged( bool isSelected );
public:
        SearchUI( GWindow *, SearchClient *);
        virtual ~SearchUI();
};
```

6.23 SearchFactoryImpl Class

This is responsible for instantiating a Search object.

```
class SearchFactoryImpl : public SearchFactory
{
public:
        SearchFactoryImpl(){}
        virtual Search * makeSearch( Window *, SearchClient * );
};
```

6.24 SearchImpl Class

This provides the implementation of the search functionality for VICI.

```
class SearchImpl : public Search
{
protected:
        GWindow *window;
        SearchClient *client;
public:
        SearchImpl( Window *, SearchClient *);
        ~SearchImpl();
        virtual void show();
};
```

Appendix A